January 1982

(4)

Report. No. STAN-CS-82-892

AFOSR-TR- 82-0646

An Algorithm for Reducing Acyclic Hypergraphs

by

Gabriel M. Kuper

Department of Computer Science

Stanford University
Stanford, CA 94305

REPORT DOCUMENTATION PAGE		**READ INSTRUCTIONS BEFORE COMPLETING FORM**
1. REPORT NUMBER AFOSR-TR- 82-0646	2. GOVT ACCESSION NO. AD-A118535	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) AN ALGORITHM FOR REDUCING ACYCLIC HYPERGRAPHS		5. TYPE OF REPORT & PERIOD COVERED TECHNICAL
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Gabriel Kuper		8. CONTRACT OR GRANT NUMBER(s) AFOSR-80-0212
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Computer Science Stanford University Stanford CA 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS PE61102F; 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Directorate of Mathematical & Information Sciences Air Force Office of Scientific Research Bolling AFB DC 20332		12. REPORT DATE JAN 1982
		13. NUMBER OF PAGES 9
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The report gives a description of an algorithm to comput efficiently the Graham reduction of an acyclic hypergraph with sacred nodes. To apply the algorithm we must already have a tree representation of the hypergraphs, and therefore it is useful when we have a fixed hypergraph and wish to compute Graham reductions many times, as we do in the System/U query interpretation algorithm.

An Algorithm for Reducing Acyclic Hypergraphs

GABRIEL KUPER[†]
Stanford University
Stanford, Calif.

Abstract

The following is a description of an algorithm to compute efficiently the Graham reduction of an acyclic hypergraph with sacred nodes. To apply the algorithm we must already have a tree representation of the hypergraphs, and therefore it is useful when we have a fixed hypergraph and wish to compute Graham reductions many times, as we do in the System/U query interpretation algorithm.

† **Work supported by grant AFOSR-80-02[illegible]2**

§1 Introduction

In System/U (see [KU]), the database is regarded as a set of ***objects*** and a number of collections of objects, called ***maximal objects*** (see [MU1]). These maximal objects can be regarded as ***hypergraphs***, where the nodes are the attributes of the database and the edges are the objects. Each maximal object is assumed to be an acyclic hypergraph. The database designer specifies the objects and maximal objects, and a tree representation for each hypergraph is then computed. To process a query on the database, the query interpreter first converts it into some combination of queries, each of involves only one maximal object. Each such query is a projection applied to the join of all objects in the maximal object, and can now be optimized using ***tableaux optimization*** (see [ASU]). [ASU] also present an optimization algorithm for ***simple*** tableaux which includes tableaux derived from hypergraphs. As noted in [MU2], optimization of tableaux derived from hypergraphs is equivalent to ***Graham reduction*** of the corresponding hypergraph with a given set of ***sacred*** nodes. The following is an algorithm to compute such a reduction, which is more efficient than applying the [ASU] algorithm mentioned above. The algorithm requires that we already have a tree ***representing*** the hypergraph. Such a representation can be constructed by carrying out an ordinary Graham reduction on the hypergraph (see [BFMY]). The tree representation is constructed once and for all when defining the database, and it is then used to guide the algorithm performing the Graham reduction with sacred nodes.

§2 Definitions

A ***hypergraph*** G is a set of ***nodes*** N, and a set of edges E, where ***edges*** are sets of nodes. We will be interested only in ***acyclic*** hypergraphs. These are defined in, e.g., [BFMY]. We shall not repeat the definition here, as it is never used here. We shall use instead two equivalent properties described below.

Definition: Let G be an acyclic hypergraph, and X a set of nodes in G. The ***Graham reduction*** of G with ***sacred*** nodes X, denoted $GR(G,X)$ (see [MU2]), is obtained by carrying out the following steps, in any order:

(1) If A is an isolated node (i.e., is in only one edge) and is nonsacred (i.e., is not in X), then delete A.

(2) If R, S are two edges in the hypergraph, such that $R \subseteq S$, then delete R.

$GR(G,X)$ consists of those nodes and edges that remain when no further reductions can be made.

Definition: Let G be an acyclic hypergraph. A tree T whose nodes are the edges of G ***represents*** G, iff for all edges R_i and R_j in G and all nodes A of G, if R_k is on the path in T connecting R_i and R_j and A is in both R_i and R_j, then A is in R_k.

Lemma 1. ***The following three properties are equivalent:***

(1) G is acyclic.

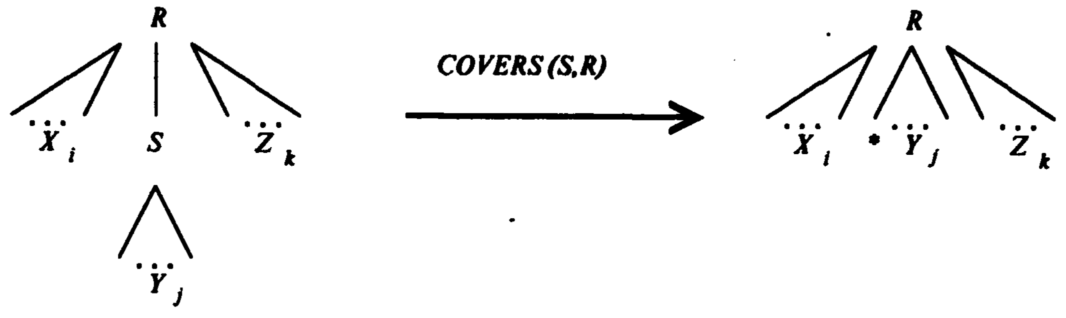

Figure 1 Deletion of S in step 1 of the algorithm.

(2) *Performing the Graham reduction on G with an empty set of sacred nodes results in the empty set.*

(3) *There exists a tree T representing the hypergraph G.* ∎

A proof can be found in [BFMY].

Definition: $COVERS(S,R)$ where S and R are edges in a hypergraph, means that R covers S, i.e.,

(1) Every isolated node in S is nonsacred.

(2) If A is a nonisolated node in S, then A is in R.

If $COVERS(S,R)$, then S can be deleted in a Graham reduction by first deleting all isolated nodes in S, and then deleting S using rule (2).

§3 An algorithm to compute GR(G,X).

Description of the Algorithm.

We are given a hypergraph G, a set of sacred nodes X, and a tree T representing the hypergraph G. Apply the following two steps to T:

1: Scan the tree from the bottom up. For each node R, examine its children from left to right. Let the current child be S, and let its children be Y_j, $j = 1,\ldots,m$ (if it has any). Also let R's children to the left of S be X_i, $i = 1,\ldots,l$, and those to S's right be Z_k, $k = 1,\ldots,n$. If $COVERS(S,R)$, carry out the transformation in Fig. 1, and continue comparing R with its children from the node marked '*', the leftmost child of S. (If S has no children then continue from Z_1.) Node R is processed either when we compare R with its rightmost child and cannot delete the child, or when the rightmost child is a leaf, and we delete it.

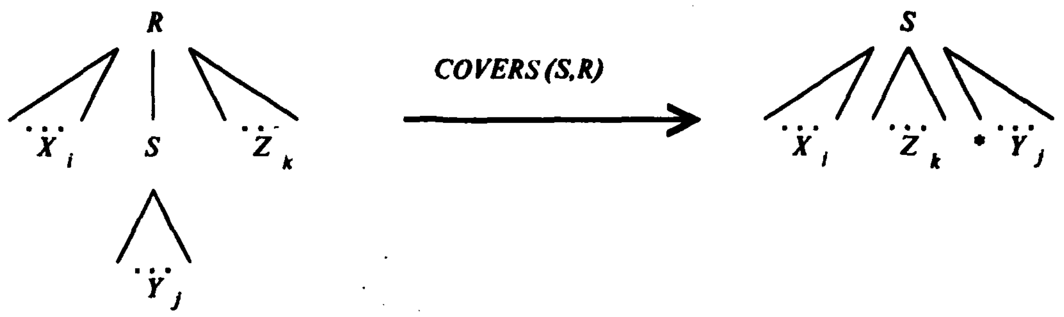

Figure 2 Deletion of R in step 2 of the algorithm.

2: Scan the tree from the top down. For each node R, examine its children from left to right. Let the current child be S, and let X_i, Y_j, Z_k be as in step 1. If $COVERS(R,S)$, carry out the transformation in Fig. 2, and continue from the node marked '*', the leftmost child of S, with R replaced by S. (We do not compare S with the Z_k's.) A node is processed either when we compare it with its rightmost child and cannot delete it, or if it is a leaf and we delete its parent.

Note that in Step. 2 we do not have to test if $COVERS(S,Z_k)$ for $k = 1,\ldots,r$, since the proof will show that this can never happen. Also note that the comparison of a node with it's children is distinct from the top-down processing of it's children, so that the Z_k's are all compared with their children.

We now prove a basic lemma required for the main theorem:

Lemma 2. *Let R, S, T be edges in a hypergraph, where $\neg COVERS(R,S)$ and $T \neq R, S$. If after deleting T, $COVERS(R,S)$ holds, then: (see Fig. 3.)*

(1) *There is a node A in the hypergraph, such that A is in R, T and in no other edge.*

(2) *In any tree representing the hypergraph, one of R and T is the parent of the other.*

Proof: Since $\neg COVERS(R,S)$ before deleting T, there must be a node A in R satisfying one of the following:

(1) A is isolated and sacred.

(2) A is nonisolated and not in S.

The first possibility cannot hold, since if it did A would remain isolated and sacred after deleting T, and therefore we would have $\neg COVERS(R,S)$ after deleting T.

Therefore the second possibility must be true, which implies that after deleting T we will still have A in R and not in S, and therefore the only way we can have $COVERS(R,S)$ is for A to become isolated upon deletion of T. Therefore, the first part of the Lemma must hold. To show that R and T are adjacent in the tree, assume they are not. Let U be any (hypergraph) edge on the path in the tree connecting them. Then A must be in U (by the definition of a tree representing a hypergraph), contradicting the first part of the Lemma. ∎

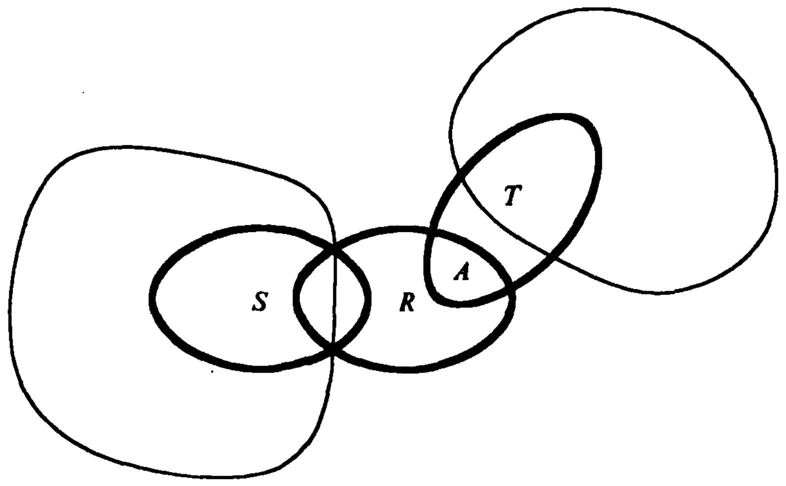

Figure 3 $COVERS(R,S)$ after deleting T.

The main theorem to be proved is:

Theorem 3. *After applying the above algorithm, we obtain a tree representing a hypergraph that can be obtained from G by applying steps in a Graham reduction with sacred nodes X. This hypergraph has the property that if U and V are two of its edges, neither of them covers the other. Therefore, if we delete all the isolated nonsacred nodes from this hypergraph, we get $GR(G,X)$.*

In order to prove the theorem we will first show two lemmas about the state of the tree after each step of the Algorithm. In their proofs we use $COVERS^1$ for the $COVERS$ relation before deleting an edge, and $COVERS^2$ afterwards.

Lemma 4. *After Step 1 of the above algorithm, we obtain a tree $T^{(1)}$ with the following properties*

(1) *The tree represents a hypergraph that can be obtained from T by a number of steps of a Graham reduction with sacred nodes X.*

(2) *If U and V are nodes in $T^{(1)}$ and U is a child of V, then $\neg COVERS(U,V)$.*

Proof: The proof is by induction on the nodes that have been processed. At each stage, assume that the tree satisfies (1). Also assume that (2) holds for all V that have been processed, and for $V = R$ and U to the left of S, where R and S are as in Fig. 1. We show that (1) and (2) still hold after deleting S, with the new relation $COVERS^2$, for all V that have been processed and for $V = R$ and U to the left of '*'.

In Fig. 1, R is the node we are interested in at this stage of the induction, and S the child we have compared with R and found that $COVERS(S,R)$.

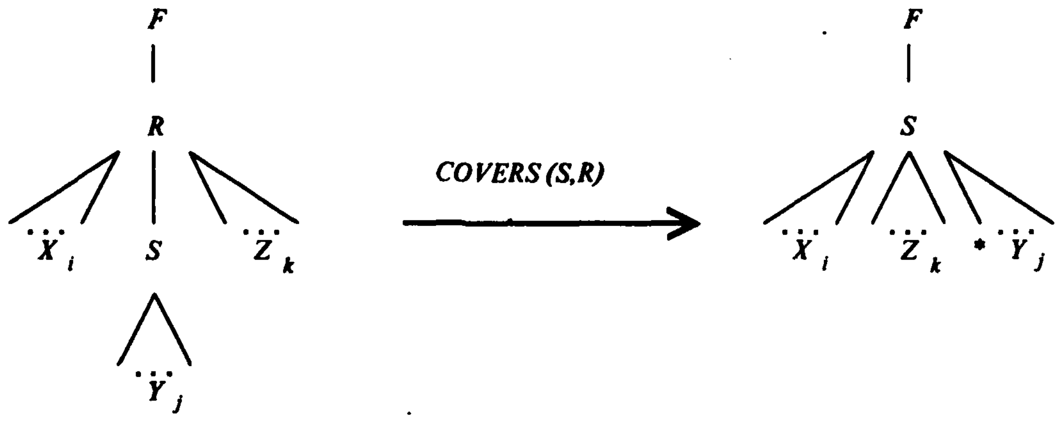

Figure 4 Deletion of R — F is its parent.

(1) Since a node S in the tree is deleted only if there is some R such that $COVERS^1(S,R)$, every deletion is a step in the Graham reduction. To show that the new tree represents its hypergraph, let U_1, U_2 be nodes in the new tree, A in $U_1 \cap U_2$, and V on a path connecting U_1 and U_2. In most cases, this is also a path in the previous tree, and therefore A is also in V. The only case when this is not true is when the path includes one or two of the Y_j's. If it only contains one of them, it can be extended to a path in the original tree by adding S to the path, and therefore A is again in V. If it connects two of the Y_j's, replace R in the path by S. If $V \neq R$, then we immediately see that A is in V. Otherwise $V = R$, in which case A in S and nonisolated (since A is in both U_1 and U_2) and $COVERS^1(S,R)$ together imply that A is in R.

(2) This will not hold after deleting S only in the following two cases:

(a) $\neg COVERS^1(U,V)$, but $COVERS^2(U,V)$, where U is a child of V.
In this case, Lemma 2 shows that U and S are adjacent. Since V is in the new tree, $V \neq S$, and since U is a child of V the only possibility is $U = R$. This implies that V is R's father, and therefore V has not yet been processed.

(b) $COVERS^2(U,V)$, where U becomes a child of V as a result of deleting S.
This can only occur when $U = Y_j$ for some j, and $V = R$. This is also a pair that has not yet been processed. ∎

Lemma 5. *After applying Step 2 to $T^{(1)}$, we obtain a tree $T^{(2)}$ with the following properties:*

(1) This tree represents a hypergraph that can be obtained from T by a number of steps of a Graham reduction with sacred nodes X.

(2) If U, V are nodes in the tree with U a child of V, then $\neg COVERS(U,V)$.

(3) If U, V are nodes in the tree with U a child of V, then $\neg COVERS(V,U)$.

Proof: As in the previous lemma, assume the result holds where V is either a node that has been already processed, or where V is the current node R, and U is to the left of S. We show

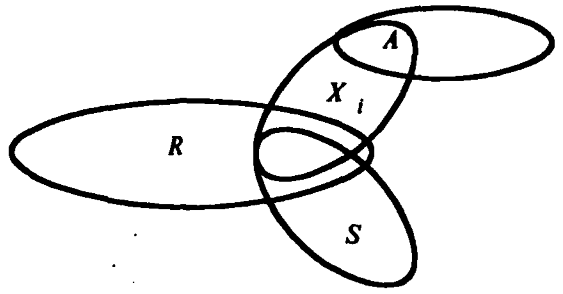

Figure 5 $\neg COVERS(X_i, S)$ after deleting R.

that the results hold after deleting R for the relation $COVERS^2$, for V that has been processed and for $V = S$ and U to the left of '*'.

In the following we shall use F to stand for R's parent, if R has one (see Fig. 4).

(1) Obviously each deletion is a step in a Graham reduction. To show that the new tree represents its hypergraph, let U_1, U_2 be nodes in the new tree, $A \in U_1 \cap U_2$, and V on a path connecting U_1 and U_2. If the path does not go through S, then the path was also a path in the tree before deleting R, and therefore $A \in V$. The path is also a path in the previous tree if it connects two Y_j's, and can be extended to one if it includes only one of the X_i's and Z_k's. The remaining case is when the path connects two of the X_i's and Z_k's. If we replace S by R we get a path in the original tree. Therefore if $V \neq S$ we immediately see that A is in V. If $V = S$ then A in R and nonisolated, and $COVERS^1(R, S)$ together imply that A is in S.

(2) Let U be a child of V, and $COVERS^2(U, V)$. As in the previous lemma there are two possibilities:

(a) U and V were not adjacent before deleting R. Then one of the following must hold:

(i) U is one of the X_i's or the Z_k's and $V = S$. Both these cases are proved in the same way, so let us take $U = X_i$. From step 1 we know that $\neg COVERS^1(X_i, R)$. Therefore there is a node A in X_i such that either A is isolated and sacred, in which case A will remain isolated and so $\neg COVERS^2(X_i, S)$, or A is nonisolated and not in R (see Fig. 5). In that case, since R is on a path in the tree connecting X_i and S, A cannot be in S. Since A is not in R, A remains nonisolated, and therefore $\neg COVERS^2(X_i, S)$.

(ii) $U = S, V = F$. From step 1, $\neg COVERS^1(S, R)$. If this is due to S containing an isolated sacred node, we immediately get $\neg COVERS^2(S, F)$. Otherwise there is a nonisolated A in S which is not in R. Then A will remain nonisolated after deleting R, and since R is on a path connecting S and F, A cannot be in F. Therefore, $\neg COVERS^2(S, F)$.

(b) U is a child of V (before deleting R), and $\neg COVERS^1(U,V)$. By Lemma 1, U must then be adjacent to R, and therefore $U =$ (i)X_i (ii) Z_k (iii) S (iv) F. The first three imply that $V = R$ which is impossible, since R has just been deleted. In the fourth case, $U = F$ and V is R's grandparent (call it G). We have $\neg COVERS^1(F,G)$, from step 1. This is due either to F containing an isolated sacred node, in which case we immediately have $\neg COVERS^2(F,G)$, or to F containing a nonisolated node A which is not in G. If $COVERS^2(F,G)$, then A must become isolated, and therefore must be in R and F only. But then, $COVERS^1(R,S)$ implies that A is in S, a contradiction.

(3) Let U be a child of V such that $COVERS^2(V,U)$. There are then two possibilities:

(a) U and V were not adjacent before deleting R. This happens when:

(i) $V = S$ and U is one of the X_i's or the Z_k's. Both cases are the same, so let us assume that $U = Z_k$. From step 1, we know that $\neg COVERS^1(S,R)$. This is due either to S containing an isolated sacred node, in which case we immediately get $\neg COVERS^2(S,Z_k)$, or to S containing a nonisolated node A which is not in R. Since A is not in R, A remains nonisolated, and since R is on a path between S and Z_k, A cannot be in Z_k. Therefore $\neg COVERS^2(S,Z_k)$.

(ii) $U = S$, $V = F$. Since the algorithm is top-down, $\neg COVERS^1(F,R)$ must already hold. If this is due to F containing an isolated sacred node, we immediately get $\neg COVERS^2(F,S)$. Otherwise, F contains a nonisolated node A that is not in R. Then A remains nonisolated, and since R is on a path between F and S, A cannot be in S. This shows that $\neg COVERS^2(F,S)$.

(b) U and V were adjacent before deleting R, with $\neg COVERS^1(V,U)$ before deleting R and $COVERS^2(V,U)$ afterwards. By Lemma 1, V and R must be adjacent and therefore V is one of these: (i)X_i (ii) Z_k (iii)S (iv)F. The first three imply that V have not yet been processed. In the fourth case, $V = F$ and U is a child of F other than R. If $\neg COVERS^2(V,U)$, lemma 1 shows that there is a node A such that A is only in F and R. But $COVERS^1(R,S)$ implies that A is in S, a contradiction. ∎

Proof of Theorem 3 By the two previous lemmas, after applying the algorithm we obtain a tree $T^{(2)}$, with the property that if U is a child of V then $\neg COVERS(U,V)$ and $\neg COVERS(V,U)$. Assume that there are edges U and V such that $COVERS(U,V)$. Let W be U's immediate successor on a path connecting U and V. Then one of the following occurs:

(a) W is a child of U. Take any A in U. If A is isolated, $COVERS(U,V)$ implies that A must be nonsacred. If A is nonisolated, then A is in V, which because the tree represents the hypergraph implies that A must be in W. Therefore $COVERS(U,W)$, contradicting Lemma 4.

(b) U is a child of W. In the same way, we get $COVERS(U,W)$, a contradiction. ∎

§4 Complexity of the Algorithm.

Let $n = |G|$ be the number of edges in the hypergraph, k the size of the largest edge in G. Both steps of the algorithm compare each node of the tree with its parent at most once (the first step does so exactly once). Each such comparison consists of two parts:

(a) Test if the *COVERS* relation holds. This requires testing if each attribute in the edge is isolated or not. If it is isolated, the *COVERS* relation does not hold if the attribute is sacred. If the attribute is nonisolated, we then have to test if the attribute is in the object above it in the tree. The *COVERS* relation will not hold if it is not. If the objects are stored as lists of attributes this requires time proportional to k. If we maintain a count of the number of edges an attribute is in, test for isolation requires constant time, and therefore testing if *COVERS* holds requires $O(k^2)$.

(b) If the *COVERS* relation holds, delete a node from the tree. If we represent the tree using pointers to left and right children and left and right siblings, the deletion requires constant time. Updating the count of edges for each attribute requires $O(k)$ time.

The two steps above have to be carried out at most twice for each edge. Therefore the complexity of the complete algorithm is $O(nk^2)$.

The algorithm in [ASU] for simple tableaux uses different data structures and so cannot be compared directly with this. It requires $O(r^4c)$ where r is the number or rows ($= n$), and c the number of columns ($\geq k$) in the tableau. If we apply the reduction algorithm using similar data structures, i.e., represent the hypergraph as an array so that testing if an attribute is in an edge requires constant time, but every attribute in the hypergraph must be examined when testing if *COVERS* holds, then (a) and (b) both require $O(c)$ time, and therefore the algorithm takes $O(rc)$ time.

References

[ASU] Aho, A.V., Y.Sagiv and J.D.Ullman, "Efficient Optimization of a Class of Relational Expressions." *ACM Transactions on Database Systems* **1:4** (1976), pp. 277-298.

[BFMY] Beeri, C., R.Fagin, D. Maier and M.Yannakakis, "On the Desirability of Acyclic Database Schemes." *RJ3131*, IBM, San Jose, 1981.

[KU] Korth, H.F. and J.D.Ullman, "SYSTEM/U:A Database System Based on the Universal Relation Assumption." *Proc. XP1 Conference*, Stonybrook, N.Y., June, 1980.

[MU1] Maier, D. and J.D.Ullman, "Maximal Objects and the Semantics of Universal Relation Databases." *TR-80-016*, Dept. of C.S.,SUNY, Stony Brook, N.Y, 1980.

[MU2] Maier, D. and J.D.Ullman, "Connections in Acyclic Hypergraphs." *Proc. ACM Symposium on Principles of Database Systems*, 1982.

FILME

-8